ENME 489Y – Remote Sensing: Spring 2018

Department of Mechanical Engineering

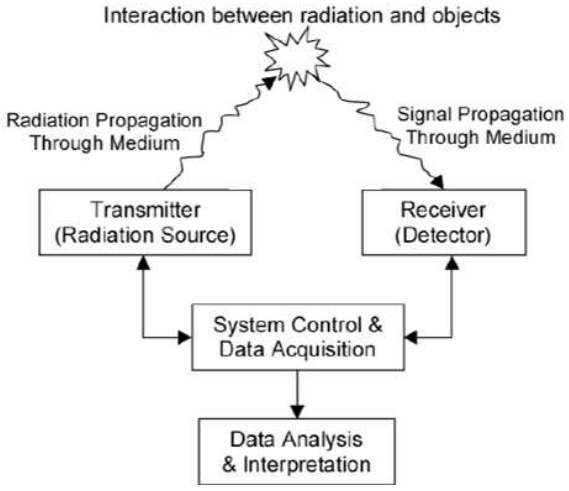
Due Date Thursday, March 15th, 2018

Submission Information
- Submit .pdf response to Question #2 via Gradescope by 9:30 am

Introduction to the triangulation lidar range measurement

Question #1 (nothing to submit)

The **transmitter** of the RPi lidar sensor consists of a LED laser, while the RPi camera is the primary component of the **receiver**. The **data acquisition** system utilizes computer vision techniques to detect the location of the laser beam as it scatters off the **target**. For each image (or frame, in the case of video), the DAQ creates and applies a color mask. For a single point laser (this assignment), the DAQ then computes the minimum enclosing circle and centroid. For a line laser (future assignments), the DAQ then identifies the pixels of the image corresponding to the line. In both instances, the (x,y) coordinates of the pixels are used to compute a range measurement based on the sensor geometry.

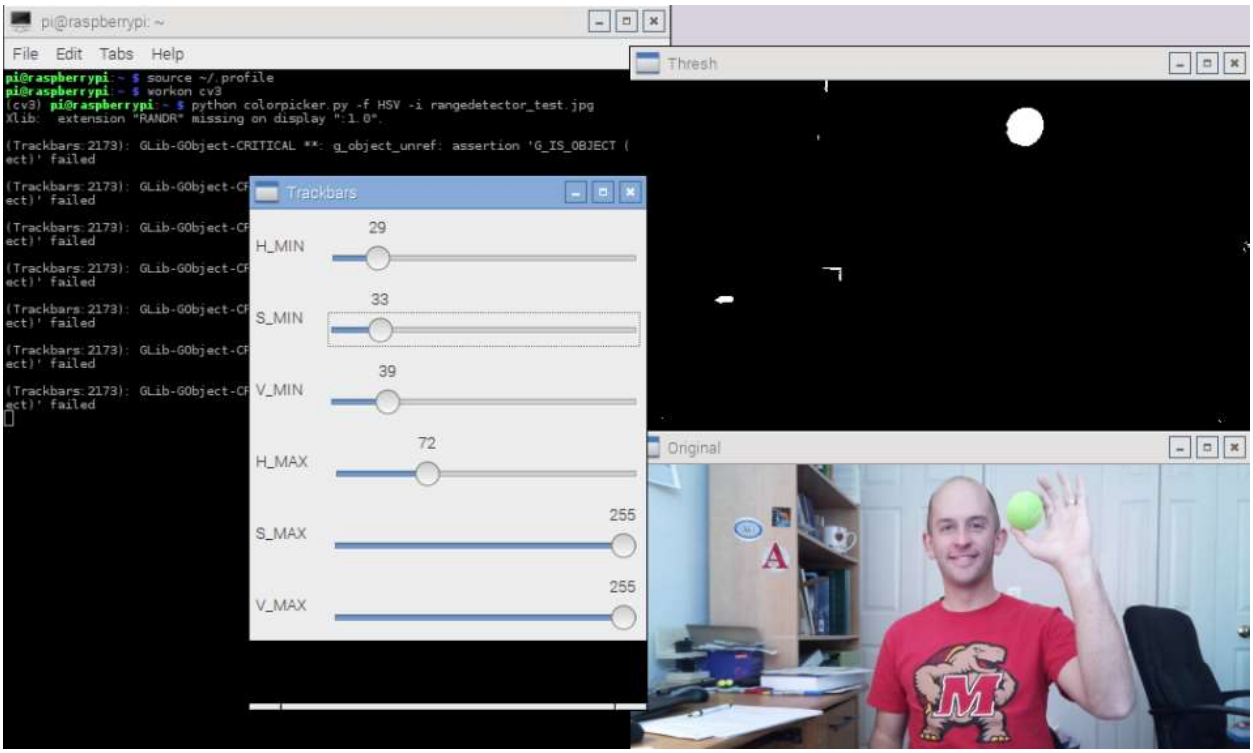


Head over to GitHub and download the Python script *colorpicker.py*.

https://github.com/oneshell/enme489y

This script has been developed to (1) input a RGB image file specified by the user (consider using your *test_image.py* script from Homework #3 to create the image file), and (2) create an HSV color mask of the image dynamically using a trackbar interface.

The following RPi screenshot illustrates the *colorpicker.py* script, in which the lower and upper boundaries of the color green in HSV color space permit detection of the green ball. The original RGB image ("Original") is shown below the HSV color mask ("Thresh"), with the Trackbars at left.



Referring back to the *stoplighttracking.py* script from Homework #3, note that the HSV mask used to track the green stoplight (and tennis ball) incorporated these lower and upper HSV value boundaries:

```
# define the lower and upper boundaries of the
# green light (circle) in the HSV color space, then initialize the
# list of tracked points
colorLower = (29, 33, 39)
colorUpper = (75, 255, 255)
```

Ensure you are able to run and utilize the *colorpicker.py* script, using any target (and colors!) of your choosing. The command line calls are provided below as well as comments in the code:

```
source ~/.profile
workon cv
python colorpicker.py -f HSV -i name_of_image_file.jpg
```

For fun, consider downloading an image of a stop sign from Google and determine the correct HSV color mask to enable ADAS detection of stop signs while driving.

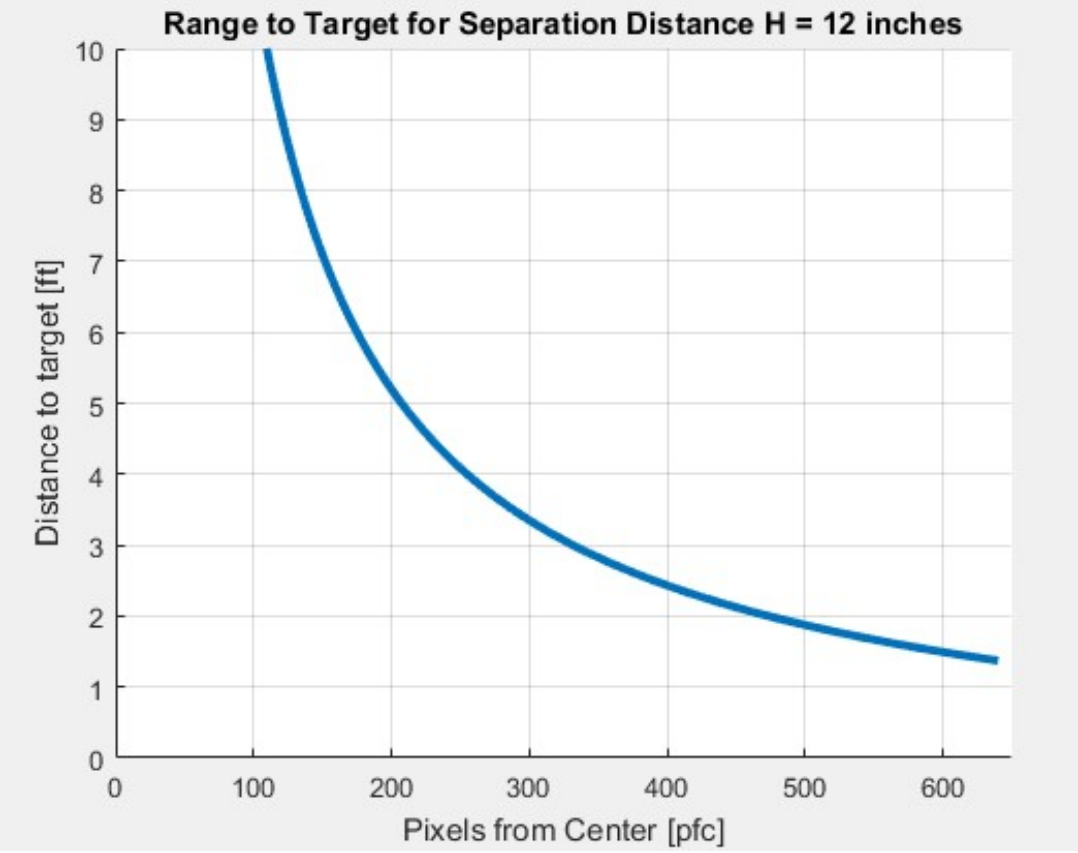
Question #2 (40 points)

Here we develop the governing theory behind the triangulation lidar range measurement, using the RPi lidar and OpenCV to track the location of the scattered laser beam (single laser beam in this assignment, line laser in future assignments). The triangulation technique is widely used in lidar and robotic vision applications. A particularly concise description of the technique has been provided by Todd Danko:

https://sites.google.com/site/todddanko/home/webcam_laser_ranger

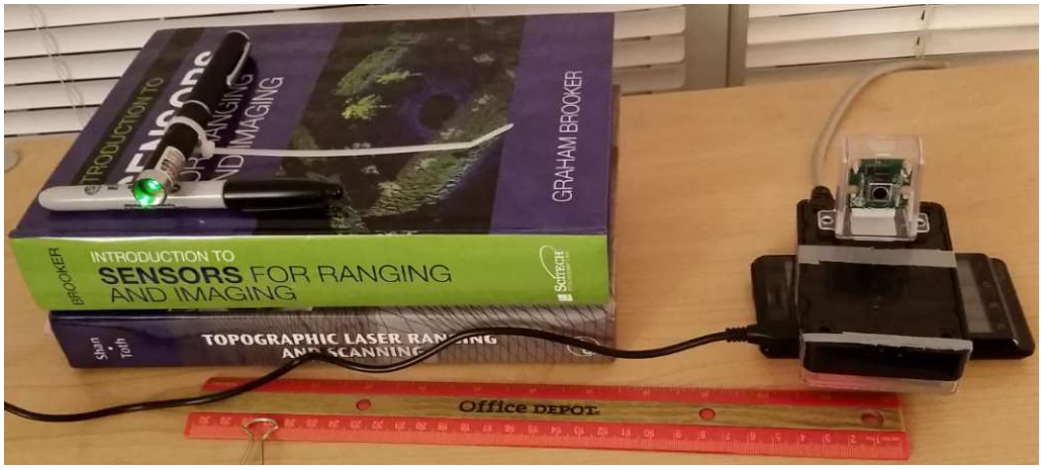
Head over to GitHub and download the Python script *test_image_laser.py*. This script outputs a single .jpg file at (x,y) dimensions of 1280 x 720 pixels, respectively, where the pixel coordinates begin at the top left corner of the image.

Step 1: Run the MATLAB and/or Python code provided in the Appendix of this assignment. The code plots the range between the lidar sensor and target based on the camera pixel corresponding to the center of the imaged laser spot, for a separation distance H of 12 inches. Physically speaking, H defines the distance between the axis of the laser and the axis of the camera, and can be adjusted as required. For this assignment, we use H = 12 inches.



Step 2: By any preferred means, setup your laser and RPi camera such that the axis of each is approximately 12 inches apart and parallel, as illustrated in the figure below.

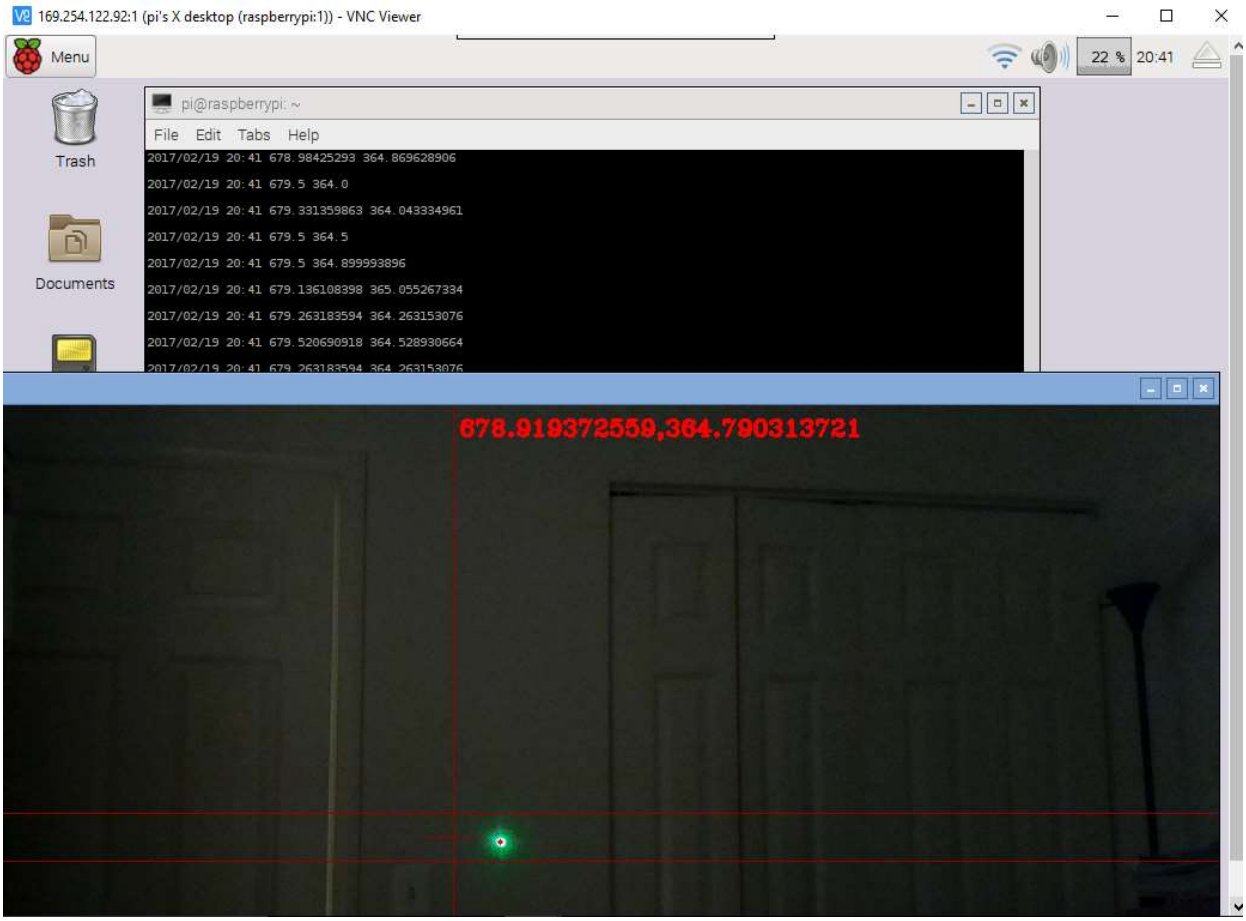
Note: ultimately, any color laser pointer will suffice for this exercise. Simply adjust the HSV mask using *colorpicker.py* as required. Dr. Mitchell has green lasers available for loan.

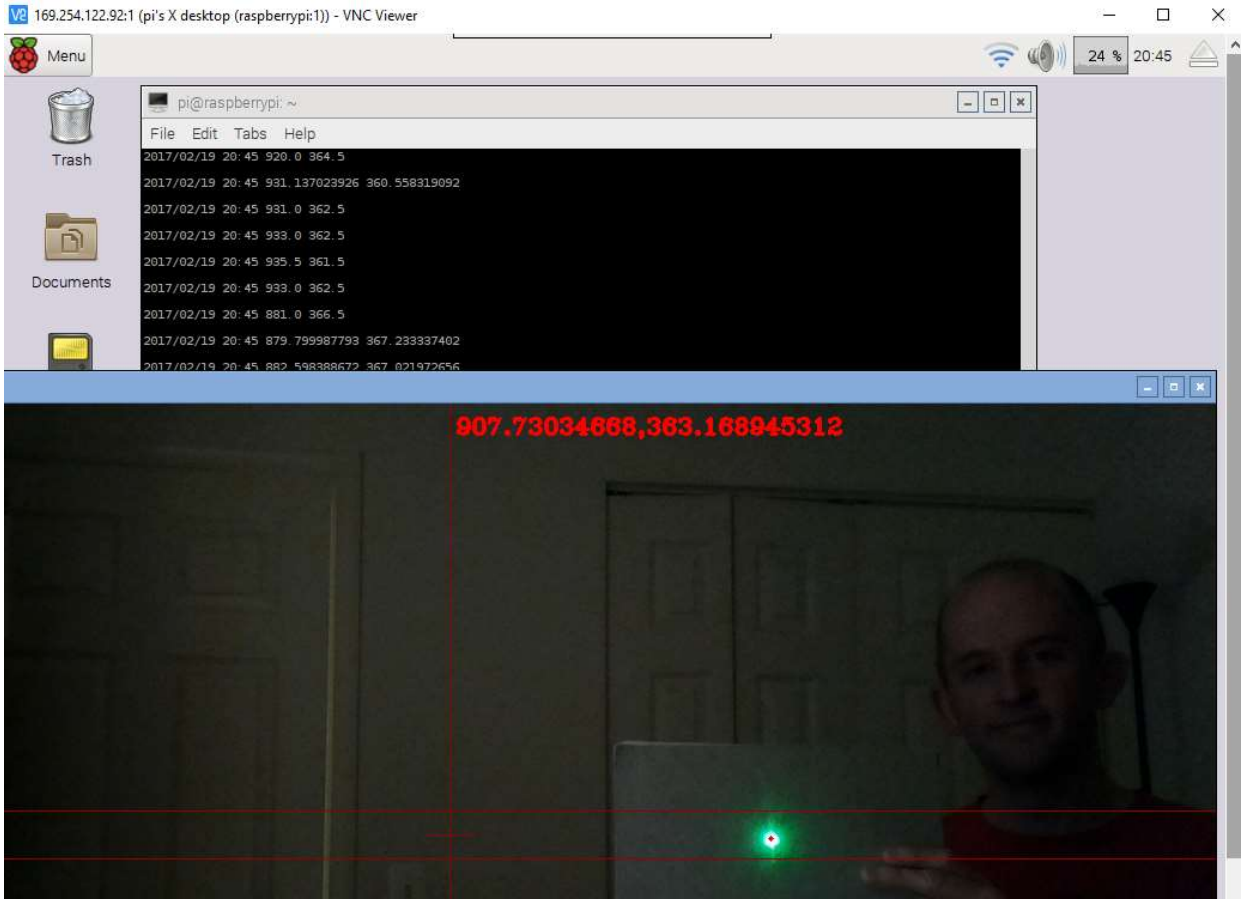


Head over to GitHub and download the Python script *lasertracker.py*. This script has been developed to (1) process video from the RPi camera, (2) utilize the object tracking code from Homework #3 to automatically identify and locate the center of the scattered laser spot, (3) plot the (x,y) coordinates of the center of the laser spot to the screen, and (4) log the date, time, and (x,y) coordinates in the text file *laserlog.txt*.

Note: the log picks up where it left off each time the script is executed, so your previous data should be safe each time you execute the script.

An illustration of the *lasertracker.py* script is provided below, for two target ranges, using the RPi VNC connection. The screenshot demonstrates the Python camera interface and the data logged during execution of the script. Notice that as the target moves further away from the lidar, the imaged laser spot moves closer to the center of the image.


169.254.122.92:1 (pi's X desktop (raspberrypi:1)) - VNC Viewer
Menu
22 %
20:41
Trash
Documents
pi@raspberrypi: ~
File Edit Tabs Help
2017/02/19 20:41 678.98425293 364.869628906
2017/02/19 20:41 679.5 364.0
2017/02/19 20:41 679.331359863 364.043334961
2017/02/19 20:41 679.5 364.5
2017/02/19 20:41 679.5 364.899993896
2017/02/19 20:41 679.136108398 365.055267334
2017/02/19 20:41 679.263183594 364.263153076
2017/02/19 20:41 679.520690918 364.528930664
678.919372559,364.790313721



Step 3: Run the *lasertracker.py* script and log data while ranging to the target of your choosing, for known distances from 0 to 10 feet, in increments of 6 inches. Plot the range data on top of the theoretical output (using the MATLAB and/or Python code provided in the Appendix), adjusting the values of *ro* and *rpc* as desired. ***Paste your combined plot into a .doc file, generate 1-2 paragraphs describing how well your data aligns with the theoretical values, and upload a .pdf of the file to Gradescope.***

Congratulations! You've now **built your own lidar sensor!**

Note 1: you'll likely need to run the *colorpicker.py* script to identify the upper and lower HSV bounds of the laser color, accounting for lighting conditions.

Note 2: consider **recording a video clip(s)** of yourself ranging to the target, showing how the imaged laser spot translates across the camera screen w.r.t. target range, **to use in your semester project video**.

Note 3: to assist the computer vision software, consider mechanically blocking the left half of the camera frame with a piece of cloth, a block, etc. This will limit the camera's field of view and should improve the performance of the computer vision algorithm.

Thinking ahead: this technique works well to measure range, but provides ***only one*** range measurement per image. How could we increase the data rate? More on this in the next assignment…

Appendix: MATLAB Code

```
% ENME 489Y: Remote Sensing
% Spring 2018
% Triangulation lidar code

clear all; close all; clc; format compact

% Refer to Todd Danko's site for details, including physical layout
% https://sites.google.com/site/todddanko/home/webcam_laser_ranger

% Define ro and rpc, which can be tweaked down the road
% Radian offset, which compensates for alignment errors
ro = -0.01
% Radians per pixel pitch, or Gain [rad/pixel]
rpc = 0.001

% Define the full span of pixels from center
% Since our Python code sets the camera frame (x,y) coordinates
% as (1280, 640), the imaged laser spot is free to translate through
% (1280/2) = 640 pixels from the center of the image
pfc = flip([0:2:640]);

% Separation distance between axes of laser pointer and webcam [cm]
% 12 inches = 0.3048 meters
H = 0.3048;

% Determine the distance to the target, given calibrated system parameters
% and pfc array evaluated from data
D = [];
for i = 1:length(pfc)
    D(i) = H/( tan(pfc(i)*rpc + ro) );
end

figure(1); hold on
% Plot distance to target as function of pfc value
% First convert H into inches and D into feet for analysis
H = 39.37*H; D = 3.28*D;
plot(pfc,D,'LineWidth',3)
xlabel('Pixels from Center [pfc]')
ylabel('Distance to target [ft]')
title('Range to Target for Separation Distance H = 12 inches')
grid
axis([0 650 0 10])
```

Appendix: Python Code

```
# ENME 489Y: Remote Sensing
# Triangulation lidar code

import numpy as np
import matplotlib
import matplotlib.pyplot as plt

# Refer to Todd Danko's site for details, including physical layout
# https://sites.google.com/site/todddanko/home/webcam_laser_ranger

# Define ro and rpc, which can be tweaked down the road
# Radian offset, which compensates for alignment errors
ro = -0.01

# Radians per pixel pitch, or Gain [rad / pixel]
rpc = 0.001

# Define the full span of pixels from center
# Since our Python code sets the camera frame (x,y) coordinates
# as (1280, 640), the imaged laser spot is free to translate through
# (1280/2) = 640 pixels from the center of the image
pfc = np.arange(0,640,2)
pfc = np.flip(pfc,0)

# Separation distance between axes of laser pointer and webcam [cm]
# 12 inches = 0.3048 meters
H = 0.3048

# Determine the distance to the target, given calibrated system parameters
# and pfc array evaluated from data
D = np.empty((0))
for i in range(pfc.shape[0]):
    D = np.append(D, H/( np.tan(pfc[i]*rpc + ro) ))

# Convert H into inches and D into feet for analysis
H = 39.37*H
D = 3.28*D

# Plot distance to target as function of pfc value
plt.figure(1)
plt.plot(pfc, D, 'b-', linewidth=3)
plt.title('Range to Target for Separation Distance H = 12 inches')
plt.xlabel('Pixels from Center [pfc]')
plt.ylabel('Distance to target [ft]')
plt.axis([0, 650, 0, 10])
plt.grid()
plt.show()
```